Design and implement closely reletade systems

Reuse of common elements

Domain Eingineering to fin and implement commmon features

Application engineering to make the individual applications.

Several definitions, common purpose is to provide reuse amongst similar applications.
Existing or new system - analyzis

Domain
A set of problems or functions that applications in that domain can solve
Could be used about:
-business area
-collection of problems
-collection of applications
-area of knowledge with common terminology

To determine a domain one could define:

what is inside
the boundaries
what is outside

Two ways of considering domains
Object oriented; see domain as real world - applications are not considered - similar to conceptual modeling -
Domain as a set of systems - closer to product-line engineering

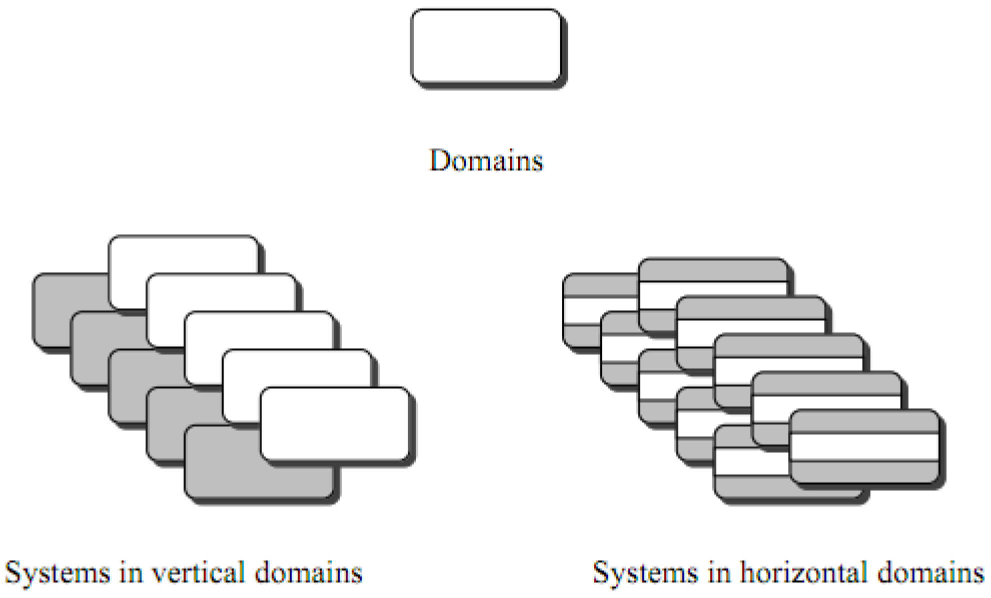
One application does not necicarilly cover a domain, and can cover parts of several, and vice-versa.

Vertical and horizontal domains:

In vertical domains, software systems are classified according to the busines area. Such systems are, for example, airline reservation systems, medical record systems

In horizontal domains, parts of a software system are classified according to their functionality. Examples are database systems, container libraries, workflow systems

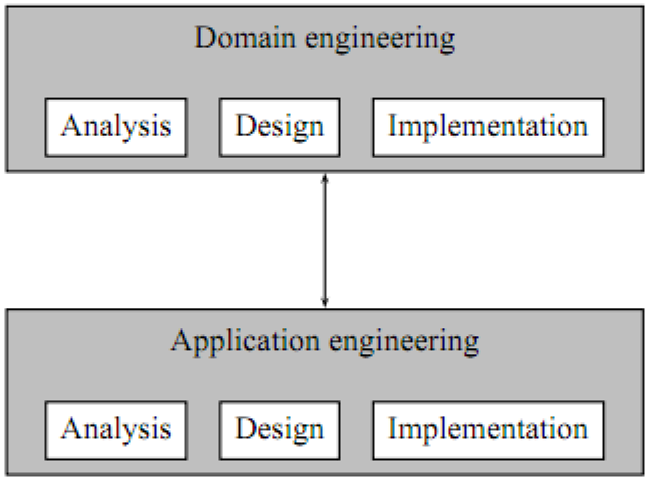
vertical domains are domains of systems, while horizontal domains are domains of parts of systems.



Domain engineering and application engineering -> product line engineering

Domain engineering and application engineering can be called engineering-forreuse and engineering-with-reuse, respectively.

can also be called assets engineering and product engineering



Domain analysis is associated with reuse, its purpose is to capture information involved with the domain to be reused in developing further applications of the same domain.

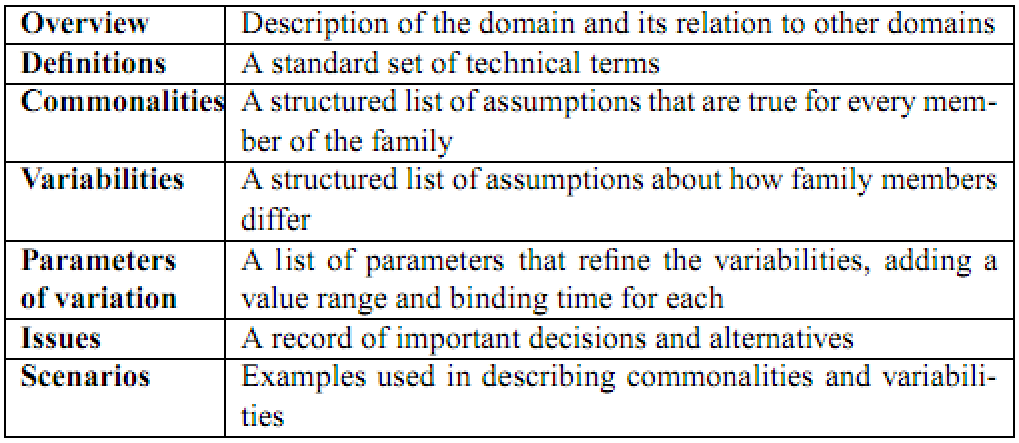
The output of domain analysis is domain model:

• domain scoping (domain definition, context analysis)
finds out the boundaries of the domain

• commonality analysis
considers both the commonalities and variabilities of the application in the domain.

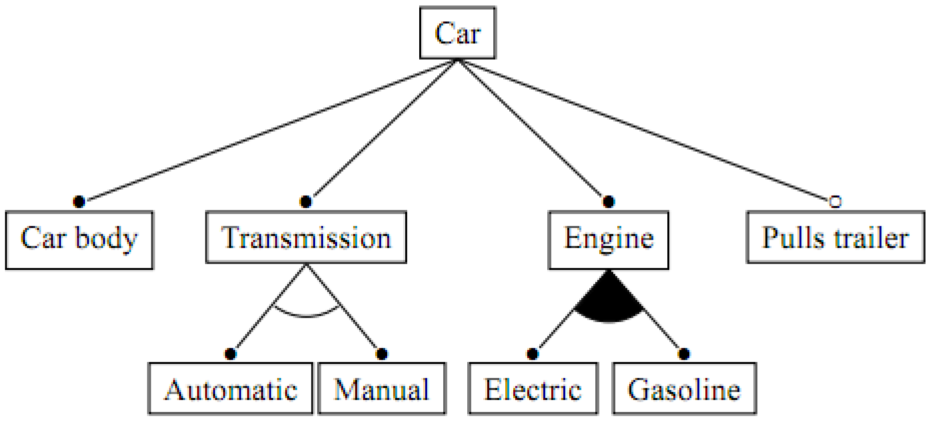
• domain dictionary (domain lexicon)
provides and defines the terms concerning the domain.

• notations (concept modeling, concept representation)
provide a uniform way to represent the concepts used in domain modeling. (data-flow charts)



Overview	Description of the domain and its relation to other domains
Definitions	A standard set of technical terms
Commonalities	A structured list of assumptions that are true for every member of the family
Variabilities	A structured list of assumptions about how family members differ
Parameters of variation	A list of parameters that refine the variabilities, adding a value range and binding time for each
Issues	A record of important decisions and alternatives
Scenarios	Examples used in describing commonalities and variabilities

• requirements engineering (feature modeling)
comprises gathering, defining, documenting, verifying, and managing the requirements that specify the applications in the domain (features)
Feature diagram of a simple car:



Note that, domain analysis can also be called as domain modeling referring to the output i.e. domain model.

Domain design means designing the core architecture for a family of applications.
The core architecture should also provide variability between applications. In this phase, it is decided how to enable this variability or configurability.

Evaluating the architecture - the sooner one discover problems, the easier to fix

Domain implementation covers the implementation of the architecture, components, and tools designed in the previous phase. This comprises, for example, writing documentation and implementing domain-specific languages and generators.

Note that domain engineering is a continuous process. The knowlegde concerning the domain should be maintained and updated all the time according to new experience, scope broadening, and new trends.

Distinction of related terms:

The terms ”domain” and ”product line” are very close to each other. However, the difference is that a domain consists of conceptual items, while a product line comprises concrete products or applications to be developed.

Requirements engineering can be divided into subtasks such as requirements elicitation to discover and understand the user’s needs, requirements analysis to refine the user’s needs, requirements specification to document the user’s needs, requirements verification to ensure that the system requirements are complete, correct, and consistent, and requirements management to schedule and coordinate the above activities concerning requirements

2.5 Domain engineering methods
This subsection introduces some of the most famous domain engineering methods. Most of the methods have originally concentrated on purely domain engineering. Later they have been extended or connected to other methods to cover the whole product-line engineering process.

FODA (Feature-Oriented Domain Analysis) considers the features of
similar applications in a domain. Features are capabilities of the applications considered from the end-user’s point of view. Features cover both common and variable aspects among related systems. They are divided into mandatory (or common), alternative, and

optional features. FODA includes different analyses such
as requirements analysis and feature analysis. It produces a domain model covering the differences between related applications. FODA is currently a part of
the model-based approach of domain engineering [Sof00a]. This approach covers both domain engineering and application engineering. The domain engineering part consists of domain analysis (FODA), domain design, and domain implementation. In addition, FODA is further extended into FORM (Feature-Oriented
Reuse Method) [KKLL99] to include also software design and implementation phases. FORM covers both analysis of domain features and using these features
to develop reusable domain artifacts.

ODM (Organization Domain Modeling) mainly concentrates on the
domain engineering of legacy systems. However, it can be applied to the requirements for new systems. ODM is tailorable and configurable, and it can be
integrated with other software engineering technologies. It combines different artifacts such as requirements, design, code, and processes from several legacy systems into reusable common assets. ODM is supported by DAGAR (Domain Architecture-based Generation for Ada Reuse) [KS96]. DAGAR process does not cover domain modeling. Thus, it applies ODM or other methods for this purpose. Instead, DAGAR process includes activities both for domain engineering and application engineering.

RSEB (Reuse-Driven Software Engineering Business) [JGJ97] is a systematic, model-driven reuse method. It composes sets of related applications from sets of reusable components. RSEB uses UML [RJB99] to specify application systems, reusable component systems, and layered architectures. Variabilities between systems are expressed with variation points and attached variants. FeatuRSEB (Featured RSEB) [GFd98] connects features (from FODA) with RSEB. Actually, FODA and RSEB have much in common. Both of them are model-driven methods providing several models corresponding the different view points of the domain. Thus, they are compatible with each other.

PuLSE (Product Line Software Engineering) divides product-line life
cycle into three parts. In the initialization phase, PuLSE is customized to fit
the particular application. Adaptation is affected by the nature of the domain,
the project structure, the organizational context, and the reuse aims. In the second phase, product-line infrastructure is constructed. This step includes scoping, modeling, and architecting the product line. In the third phase, the productline infrastructure is used to create individual products. This concerns instantiating the product-line model and architecture. Each of these phases is associated
with product-line infrastructure evolution. Each phase should consider changing requirements and changing concepts within the domain. PuLSE has several components. PuLSE-DSSA (PuLSE - Domain-Specific Software Architecture) [ABFG00], for example, develops a domain specific architecture based on the product-line model. As other examples, PuLSE-Eco concentrates on economic scoping, and PuLSE-EM on evolution and management.

FAST (Family-Oriented Abstraction, Specification, and Translation) process covers the whole product-line engineering process [WL99]. However, it divides domain engineering into two parts: domain analysis and domain implementation.
Thus, the issues involving domain design are considered in the domain analysis

phase. FAST provides systematic guidance to each step during product-line engineering. These steps can be proceeded as transitions between states. FAST also describes which artifacts are produced in each phase.

Object-oriented analysis and design (OOA/D) methods are widely used in software engineering. However, they mainly concentrate on single system analysis and design. Object-orientation was first thought to inherently bring reuse, however, nowadays it has been agreed that to enable reuse, it has to be invested and planned beforehand.

However, OOA/Ds do not support reuse, and they do not guide in designing families of applications. In [CE00], the problems OOA/D methods have been listed as follows:

- no distinction between domain engineering and application engineering
- no domain scoping phase
- no differentiation between modeling variability within one application and between several applications
- no implementation-independent means of variability modeling.

EMERGING PROBLEMS

Non-established terminology
is mainly due to the immaturity of the research area (considered throughout the report).

Incompleteness of domain engineering
means that domain modeling can never be complete, instead, the model can always be made more accurate. In addition, domain knowledge from different sources can be inconsistent with each other (considered in Subsection 2.3.3).

Distributed domains
are typically associated with domain engineering of existing applications, i.e. reengineering aspects (considered in Subsections 2.1.2 and 2.1.3).

Overlapping domains
are due to the difficulties in determining the borders of domains or in dividing domains into subdomains (considered in Subsection 2.1.1).

Lacking support for domain engineering in well-accepted object modeling techniques
is due to the immaturity of the research area of domain engineering and to the belief that object orientation inherently makes applications reusable (considered in Subsection 2.6).

The last problem is probably the most severe one of the above problems. In addition to object modeling techniques, the similar deficience concerns the modeling language UML. It does not support domain engineering, neither. However, the technique concerning MDA (Model Driven Architecture) that is based on UML, may provide support for domain engineering, too [dMJS02, Arc01]. Thus, that topic is an important subject for further research.